Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

# РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

## *К КУРСОВОЙ РАБОТЕ*

## *НА ТЕМУ:*

## *«Симуляция движения сфер под действием собственного гравитационного поля»*

Студент ИУ7-56Б (Группа) _____ (Подпись, дата) Куликов Е. А. (И. О. Фамилия)

Руководитель курсовой работы _____ (Подпись, дата) Клорикьян П. В. (И. О. Фамилия)

*2024 г.*

# РЕФЕРАТ

Расчетно–пояснительная записка, 38с., 13 рис., 6 ист., 1 прил.

КОМПЬЮТЕРНАЯ ГРАФИКА, СИМУЛЯЦИЯ ГРАВИТАЦИИ, АЛГОРИТМ Z–БУФЕРА, ЗАКРАСКА, МОДЕЛЬ ОСВЕЩЕНИЯ, ПОСТРОЕНИЕ ТЕНЕЙ.

Целью курсовой работы является реализация программы для моделирования движения группы сферических тел в пространстве под действием их собственного гравитационного поля.

Для визуализации использовался алгоритм Z–буфера, для представления объектов сцены использовалась поверхностная модель.

В процессе работы проанализированы различные алгоритмы, методы представления объектов, методы закрасок и модели освещения. Выбраны подходящие для решения поставленной задачи. Выбранные алгоритмы реализованы в виде одной программы с пользовательским интерфейсом.

Проведено исследования быстродействия программы в зависимости от количества идентично освещенных объектов одинакового размера на сцене. Из результатов следует, что время отрисовки увеличивается как при увеличении количества полигонов, которыми представлены объекты, так и при увеличении количества объектов.

# Содержание

# ВВЕДЕНИЕ

Исследование гравитационного взаимодействия тел имеет фундаментальное значение для понимания происходящих в космосе процессов, таких как формирование планет и их спутников, эволюция звезд и образование вокруг них планетных систем, выявление гравитационных аномалий для поиска полезных ископаемых. Моделирование этих процессов является сложной задачей, особенно при большом количестве одновременно взаимодействующих тел, и требует больших вычислительных мощностей и эффективных алгоритмов.

Целью курсовой работы является реализация программы для моделирования движения группы сферических тел в пространстве под действием их собственного гравитационного поля. Для достижения поставленной цели необходимо решить ряд задач:

— Провести анализ существующих алгоритмов компьютерной графики;

— Выбрать подходящие для данной работы алгоритмы;

— Выбрать язык программирования и среду разработки;

— Реализовать выбранные алгоритмы в виде одной программы с графическим интерфейсом;

— Провести исследование быстродействия разработанного ПО в зависимости от количества объектов сцены.

# 1 Аналитическая часть

В данном разделе будет проведен анализ существующих алгоритмов построения реалистических изображений и выбор подходящих алгоритмов для решения поставленной задачи.

Задача симуляции движения тел под действием их собственной гравитации будет решаться как серия подзадач с временным интервалом $\delta t$ между ними — состояний сцены в моменты времени, кратные $\delta t$. При достаточно малом $\delta t$ симуляция будет достаточно плавной и реалистичной. Каждая подзадача состоит из подпунктов:

— для каждого состояния сцены требуется расчитать суммарную гравитационную силу, действующую на каждое тело сцены;

— используя вычисленные силы расчитать ускорения тел, затем рассчитать прирост скорости каждого тела и его переменещние за промежуток $\delta t$;

— определить столкновения тел и считать их абсолютно неупругими, то есть столкновениями со слипанием;

— отрисовать полученное состояние сцены.

Для решения данной задачи необходимы: оптимальный формат хранения данных об объектах сцены, достаточно быстрый для симуляции в реальном времени алгоритм отрисовки, быстрая но в то же время реалистичная закраска сфер вкупе с моделью освещения. Также требуется разработать алгоритм расчета положения каждого тела в каждом конкретном состоянии сцены, алгоритм определения столкновения тел и алгоритм расчета параметров нового тела, образовавшегося при столкновении других тел.

## 1.1 Способ задания трехмерного объекта

В компьютерной графике для представления трехмерного объекта используются модели. Существуют три основных типа моделей: каркасная, поверхностная и твердотельная. Каждая из данных моделей имеет свои плюсы и минусы, поэтому в данной секции требуется выбрать оптимальную для поставленной задачи модель.

### 1.1.1 Каркасная модель

Каркасная модель представляет собой совокупность вершин и ребер отображаемого объекта. Каркасное моделирование представляет собой моделирование самого низкого уровня и имеет ряд серьезных ограничений, большинство из которых возникает из-за недостатка информации о гранях, заключенных между ребрами и вершинами, и невозможности выделить внешнюю и внутреннюю область изображения твердого объемного тела. Однако каркасная модель требует гораздо меньше компьютерной памяти, чем две другие модели, и может оказаться вполне пригодной для решения некоторых задач, в которых требуются простые формы объектов. Неоднозначность — основной недостаток этой модели. Невозможно однозначно интерпретировать видимость или невидимость граней моделируемого тела. Поэтому невозможно использовать вместе с каркасными моделями систему освещения и применять к ним алгоритмы удаления невидимых граней [1].

### 1.1.2 Поверхностная модель

Поверхностное моделирование определяется в терминах точек, линий и поверхностей. При построении поверхностной модели предполагается, что объекты ограничены поверхностями, которые отделяют их от окружающей среды. Эта оболочка изображается графическими поверхностями. Поверхность технического объекта снова становится ограниченной контурами, но эти контуры уже являются результатом 2-х касающихся или пересекающихся поверхностей. Точки объектов - вершины, могут быть заданы пересечением трех поверхностей. В основу поверхностной модели положены два основных математических положения:

— Любую поверхность можно аппроксимировать многогранником, каждая грань которого является простейшим плоским многоугольником;

— Наряду с плоскими многоугольниками в модели допускаются поверхности второго порядка и аналитически неописываемые поверхности, форму которых можно определить с помощью различных методов аппроксимации и интерполяции.

В отличие от каркасной модели, поверхностные модели имеют внутреннюю и внешнюю части [1].

### 1.1.3 Твердотельная модель

Данная модель отличается от поверхностной модели тем, что дополнительно использует информацию о том, с какой стороны поверхности расположен материал. Обычно для этого хранится вектор внутренней нормали к поверхности модели. Данная модель предоставляет наиболее полное описание трехмерного объекта в программе, однако, в некоторых случаях эта модель избыточна.

### 1.1.4 Выбранная модель

Для решения поставленной задачи не подойдет каркасная модель, так как она не предоставляет информации о поверхностях объектов, только об их форме, что не позволит эффективно использовать модель освещения и алгоритмы удаления невидимых ребер, а значит не позволит построить реалистическое изображение. Твердотельная модель не подходит для решения задачи, так как требует хранения лишней информации о расположении материала объекта, которая не важна для пользователя. Поверхностная модель подходит для решения задачи так как одновременно позволяет построить реалистическое изображение, а также не требует хранения лишней информации.

## 1.2 Описание объектов трехмерной сцены

Трехмерная сцена состоит из следующих объектов:

— камера, которая задается тремя векторами — вектором положения камеры (означает точку в пространстве, в которой расположена камера), вектором направления взгляда, вектором вертикали (означает, какой вектор будет вертикальным при перспективном проецировании сцены на плоскость экрана);

— точечный источник света, который задается вектором положения в пространстве. Источник не является направленным, то есть испускает свет во все стороны;

— тело, состоящее из сфер. Каждая сфера задается радиусом, массой (которая зависит от радиуса, так как плотность у всех сфер одинаковая), также для каждой сферы строится аппроксимирующая ее поверхностная

модель, которая хранится вместе с информацией о физической сфере. Каждое тело изначально задается как одиночная сфера, но по ходу симуляции из-за абсолютно неупругих столкновений сфер(их слипания), сферы двух тел могут объединяться в одном теле.

## 1.3 Удаление невидимых линий и поверхностей

Сложность задачи удаления невидимых линий и поверхностей привела к появлению большого числа различных способов ее решения. Наилучшего решения общей задачи удаления невидимых линий и поверхностей не существует. Алгоритмы могут работать в двух пространствах — в объектном (пространство мировых координат) и в экранном (пространство экранных координат) [2]. Для решения поставленной задачи, в частности для моделирования процесса в реальном времени, скорость работы алгоритма важнее его точности, поэтому неважно в каком пространстве он будет работать.

### 1.3.1 Алгоритм Робертса

Алгоритм Робертса представляет собой первое известное решение задачи об удалении невидимых линий. Это метод, работающий в объектном пространстве [2]. Алгоритм выполняется в 4 основных этапа:

— Построение матриц всех тел на сцене. Алгоритм требует чтобы все обрабатываемые тела были выпуклыми, невыпуклые должны быть разбиты на выпуклые части. Для каждого тела составляется матрица из коэффициентов уравнений плоскостей, задающих это тело. Предполагается, что матрицы составляются таким образом, чтобы скалярное произведение любого столбца на вектор точки, находящейся внутри тела, было положительным;

— Для каждого тела удаляются ребра, экранируемые самим этим телом. Для этого рассматривается вектор взгляда наблюдателя $E = (0, 0, -1, 0)$, который умножается на матрицу тела. В полученном векторе отрицательные компоненты соответствуют невидимым граням исходного тела;

— Для каждого тела удаляются сегменты ребер, экранируемые другими телами. Для этого строится луч из точки наблюдения в точку ребра. Если этот луч встречает на своем пути преграду - другое тело, то он

проходит по положительную сторону от всех плоскостей, задающих тело (что можно определить умножив вектор на матрицу тела), а значит точка на ребре будет невидима;

— Удаляются сегменты ребер тел, связанных отношением протыкания.

Алгоритм является очень точным, так как работает в объектном пространстве, однако он также является очень медленным из-за теоретического роста сложности в $O(n^2)$ где $n$ — число объектов [2].

### 1.3.2 Алгоритм Z–буфера

Это один из простейших алгоритмов удаления невидимых поверхностей [2]. Работает в пространстве изображения. Идея Z–буфера является простым обобщением идеи о буфере кадра. Буфер кадра используется для запоминания атрибутов (интенсивности) каждого пиксела в пространстве изображения, Z–буфер — это отдельный буфер глубины, используемый для запоминания координаты z или глубины каждого видимого пиксела в пространстве изображения. В процессе работы глубина или значение z каждого нового пиксела, который нужно занести в буфер кадра, сравнивается с глубиной того пиксела, который уже занесен в Z–буфер. Если это сравнение показывает, что новый пиксел расположен впереди пиксела, находящегося в буфере кадра, то новый пиксеп заносится в этот буфер и, кроме того, производится корректировка Z–буфера новым значением [2].

Главное преимущество алгоритма — его быстродействие. Его сложность не более чем линейна из-за фиксированных габаритов экрана, также этот алгоритм тривиально решает задачу о невидимых линиях и поверхностях, а также задачу визуализации пересечений сложных поверхностей. Кроме того алгоритм легко модифицируется для учета теней с использованием теневого Z–буфера, а также для учета модели освещения [2].

Основной недостаток алгоритма — большой объем используемой памяти под Z–буфер и буфер кадра. Буфер кадра размером 512x512x24 бит в комбинации с Z -буфером размером 512x512x20 бит требует почти 1.5 мегабайт памяти [2].

### 1.3.3 Алгоритм обратной трассировки лучей

Главная идея, лежащая в основе этого метода, заключается в том, что наблюдатель видит любой объект посредством испускаемого неким источником света, который падает на этот объект и затем доходит до наблюдателя. Свет может достичь наблюдателя, отразившись от поверхности, преломившись или пройдя через нее. Если проследить за лучами света, выпущенными источником, то можно убедиться, чго весьма немногие из них дойдут до наблюдателя.

В этом алгоритме предполагается, чго сцена уже преобразована в пространство изображения. Считается, что точка зрения или наблюдатель находится в бесконечности на положительной полуоси z. Поэтому все световые лучи параллельны оси z. Каждый луч, исходящий от наблюдателя, проходит через центр пиксела на растре до сцены. Траектория каждого луча отслеживается, чтобы определить, какие именно объекты сцены, если таковые существуют, пересекаются с данным лучом. Необходимо проверить пересечение каждого объекта сцены с каждым лучом. Если луч пересекает объект, то определяются все возможные точки пересечения луча и объекта. Эти пересечения упорядочиваются по глубине. Пересечение с максимальным значением z представляет видимую поверхность для данного пиксела. Атрибуты этого объекта используются для определения характеристик пиксела.

Из-за того, что приходится искать много точек пересечения луча с объектами, алгоритм является не таким быстрым, как алгоритм Z–буфера [2].

### 1.3.4 Выбранный алгоритм

Для решения поставленной задачи лучше всего подходит алгоритм Z–буфера как самый быстрый, практически не зависимый от сложности сцены [2]. Кроме этого, алгоритм достаточно просто модифицируется для учета теней и модели освещения [2].

## 1.4 Алгоритмы закраски

Модели закрашивания используются для затенения полигонов модели под воздействием некоторого источника освещения. Уровень освещенности полигона ищется в зависимости от взаимного расположения полигона и источника света. Существует три основных способа закраски многоугольников: однотонная закраска, закраска с интерполяцией интенсивности и закраска с интерполяцией векторов нормали.

### 1.4.1 Однотонная закраска

При однотонной закраске предполагается, что и источник света, и наблюдатель находятся в бесконечности, поэтому скалярное произведение вектора нормали к поверхности и вектора падения света постоянно. Поэтому каждый полигон закрашивается одной интенсивностью. На изображении могут быть хорошо заметны резкие перепады интенсивности между различно закрашенными многоугольниками. Если многоугольники представляют собой результат аппроксимации криволинейной поверхности, то изображение недостаточно реалистично так как явно видны ребра полигонов.

### 1.4.2 Закраска по Гуро

Методом Гуро можно получить сглаженное изображение. Для того чтобы изобразить объект методом построчного сканирования, нужно в соответствии с моделью освещения рассчитать интенсивность каждого пиксела вдоль сканирующей строки. Сначала аппроксимируются нормали в вершинах многоугольника. Однако сканирующая строка не обязательно проходит через вершины. Поэтому, затем с помощью билинейной интерполяции вычисляется интенсивность каждого пиксела на сканирующей строке [2].

### 1.4.3 Закраска по Фонгу

Закраска Фонга требует больших вычислительных затрат, однако она позволяет разрешить многие проблемы метода Гуро. При закраске Гуро вдоль сканирующей строки интерполируется значение интенсивности, а при закраске Фонга — вектор нормали. Затем он используется в модели освещения для вычисления интенсивности пиксела. При этом достигается лучшая локальная

аппроксимация кривизны поверхности и, следовательно, получается более реалистичное изображение. В частности, правдоподобнее выглядят зеркальные блики. При закраске Фонга аппроксимация кривизны поверхности производится сначала в вершинах многоугольников путем аппроксимации нормали в вершине. После этого билинейной интерполяцией вычисляется нормаль в каждом пикселе [2].

Рисунок 1.1 – Сравнение закрасок однотонной, Гуро и Фонга

### 1.4.4 Выбранный алгоритм

Так как основной объект сфены — сфера, простая закраска не подойдет, будут видны ребра. Лучше всего для решения поставленной задачи подходит закраска по Гуро, так как требует меньших вычислений [2], чем закраска по Фонгу, но при этом достаточно реалистично и без ребристости передает криволинейные поверхности [2], в частности — сферы.

## 1.5 Модель освещения

Модель освещения в компьютерной графике — это метод, который используется для воспроизведения световых эффектов в визуализируемых средах. Цель модели освещения — вычислить цвет каждого пикселя или количество света, отражённого для различных поверхностей в сцене.

Свойства отраженного света зависят от строения, направления и формы источника света, от ориентации и свойств поверхности. Отраженный от объекта свет может также быть диффузным или зеркальным. Диффузное отражение света происходит, когда свет как бы проникает под поверхность объекта, поглощается, а затем вновь испускается.

Свет точечного источника отражается от идеального рассеивателя по закону косинусов Ламберта:

$$I = I_l \cdot k_d \cdot \cos(\Theta) \tag{1.1}$$

где $I$ — интенсивность отраженного света, $I_l$ — интенсивность точечного источника, $k_d$ — коэффициент диффузного отражения ($0 \leq k_d \leq 1$), $\Theta$ — угол между направлением света и нормалью к поверхности $0 \leq \Theta \leq \frac{\Pi}{2}$. При этом если угол $\Theta$ больше чем $\frac{\Pi}{2}$, то источник света находится за объектом.

На объекты реальных сцен падает еще и рассеянный свет, отраженный от окружающей обстановки, например от стен комнаты. Рассеянному свету соответствует распределенный источник, который в машинной графике заменяется на коэффициент рассеяния — константу, которая входит в формулу в линейной комбинации с членом Ламберта:

$$I = I_a \cdot k_a + I_l \cdot k_d \cdot \cos(\Theta) \tag{1.2}$$

где $I_a$ — интенсивность рассеянного света, $k_a$ — коэффициент диффузного отражения рассеянного света $0 \leq k_a \leq 1$.

Интенсивность света обратно пропорциональна квадрату расстояния от источника, и объект, лежащий дальше от него, должен быть темнее, поэтому вводится коэффициент линейного затухания:

$$I = I_a \cdot k_a + \frac{I_l \cdot k_d \cdot \cos(\Theta)}{K + d} \tag{1.3}$$

где $K$ — произвольная постоянная, $d$ — расстояние от объекта до центра проекции.

Интенсивность зеркально отраженного света рассчитывается в соответствии с моделью Фонга, тогда итоговая формула интенсивности выглядит так:

$$I = I_a \cdot k_a + \frac{I_l}{K + d} \cdot (k_d \cdot \cos(\Theta) + k_s \cdot \cos^n(\alpha)) \tag{1.4}$$

где $k_s$ — константа, угол $\alpha$ — угол между вектором наблюдения и вектором отражения, $n$ — степень, аппроксимирующая пространственное распределение зеркально отраженного света.

В машинной графике эта модель часто называется функцией закраски и применяется для расчета интенсивности или тона точек объекта или пикселов изображения. Чтобы получить цветное изображение, нужно найти функции закраски для каждого из трех основных цветов [2].

В данной работе будет использована модель освещения с учетом зеркального и диффузного отражения света источника.

## Вывод

В данном разделе были выбраны алгоритмы, подходящие для достижения поставленной цели:

— выбран поверхностный способ задания трехмерного объекта;

— выбран алгоритм Z–буфера для удаления невидимых линий и поверхностей;

— выбран алгоритм закраски по Гуро в сочетании с моделью освещения, учитывающей зеркальную и диффузную составляющую отраженного света.

# 2 Конструкторская часть

## 2.1 Расчет движения тел сцены

Каждое тело сцены представляет собой набор из $n \geq 1$ соединенных сфер с одинаковыми плотностями, то есть масса сферы зависит от ее радиуса. Моделируется движение тела сцены в вакууме под действием сил, действующих на него со стороны других тел.

Сила взаимодействия тел вычисляется по закону всемирного тяготения [3]:

$$F = G \cdot \frac{m_1 \cdot m_2}{r^2} \tag{2.1}$$

где $F$ — сила взаимного притяжения двух тел, $r$ — расстояние между телами, $m_i$ — масса i-го тела, $G$ — гравитационная постоянная. Эта сила направлена вдоль линии, соединяющей центры масс данных тел.

Векторная сила всемирного тяготения:

$$\vec{F_{21}} = G \cdot \frac{m_1 \cdot m_2}{r^3} \cdot \vec{r_{12}} \tag{2.2}$$

где $F_{12}$ — векторная сила действия второго тела на первое, $r_{12}$ — вектор, соединяющий центры масс первого и второго тела.

Гравитационное взаимодействие между телами осуществляется посредством поля тяготения, или гравитационного поля. Основное свойство поля тяготения заключается в том, что на всякое тело массой $m$, внесенное в это поле, действует сила тяготения:

$$\vec{F} = m \cdot \vec{g} \tag{2.3}$$

где $\vec{g}$ — коэфициент, не зависящий от $m$ и называющийся напряженностью поля тяготения.

Из уравнений 2.2 и 2.3 получается что напряженность поля тяготения тела массой $m$ на расстоянии $r$ равна:

$$\vec{g} = -G \cdot \frac{m}{r^3} \cdot \vec{r} \tag{2.4}$$

вектор $\vec{g}$ направлен по направлению к телу $m$ создающему данное гравитационное поле.

В силу принципа аддитивности гравитационных сил, сила действия $n$ тел на данное тело $A$ равна сумме действий каждого из $n$ тел на это тело $A$, то есть равна векторной сумме всех сил, действующих на тело. Такое же правило выполняется и для векторов напряженности гравитационного поля. Кроме того, напряженность гравитационного поля численно и по направлению равна ускорению свободного падения в этом поле [3].

Для того чтобы определить смещения тел между двумя кадрами за малое время $\delta t$ используется следующий алгоритм:

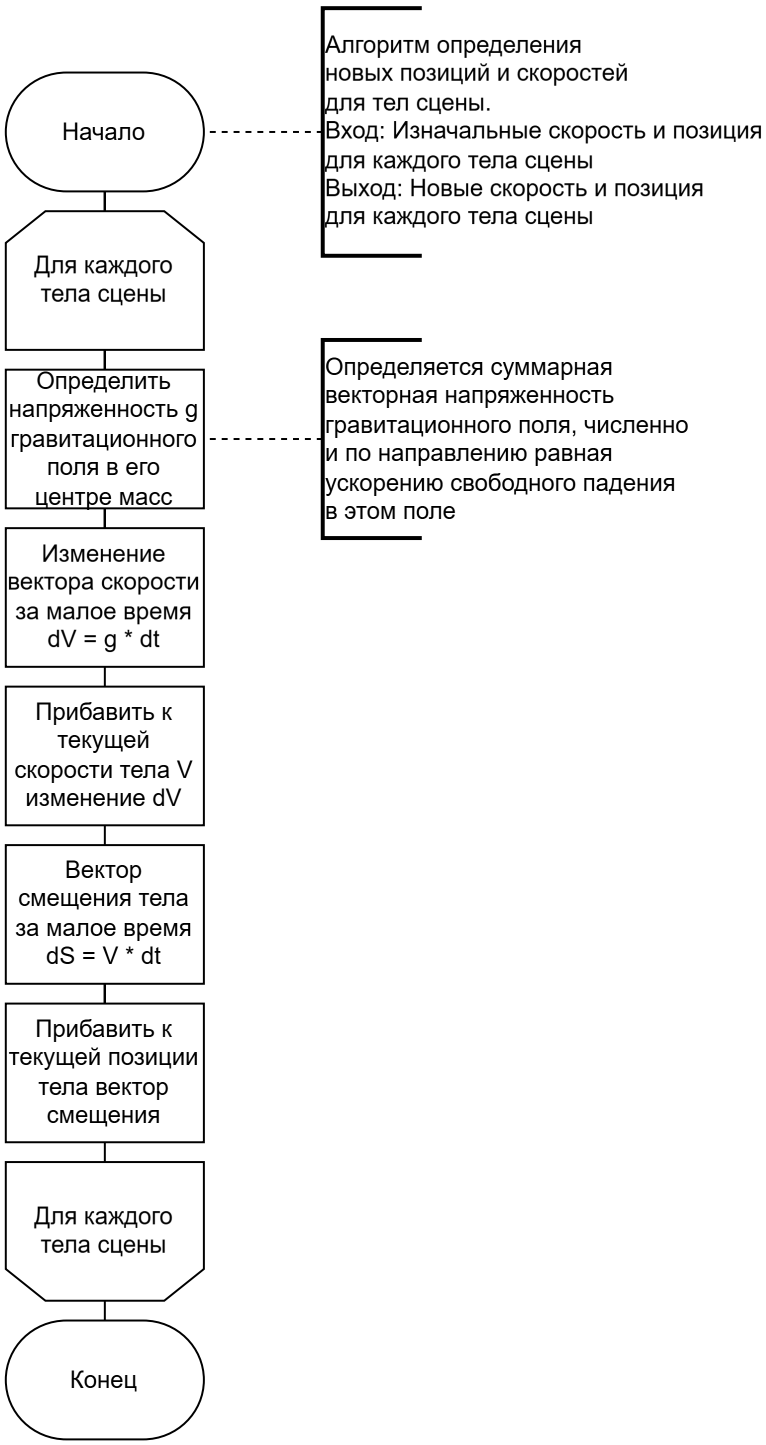


Рисунок 2.1 – Алгоритм расчета новых скоростей и позиций тел

## 2.2 Определение столкновений тел

Так как при соударении тел сцены происходит абсолютно неупругий удар, то есть тела слипаются, требуется определять столкновения тел, состоящих из нескольких сфер каждое.

Для определения столкновения двух сфер при известных координатах центров и радиусах достаточно определить, что больше: расстояние $dx$ между центрами сфер или сумма их радиусов $r_1 + r_2$. Если $dx > r_1 + r_2$ то сферы не пересекаются, иначе пересекаются.

Для определения столкновения двух составных тел и объединения их в одно используется следующий алгоритм:

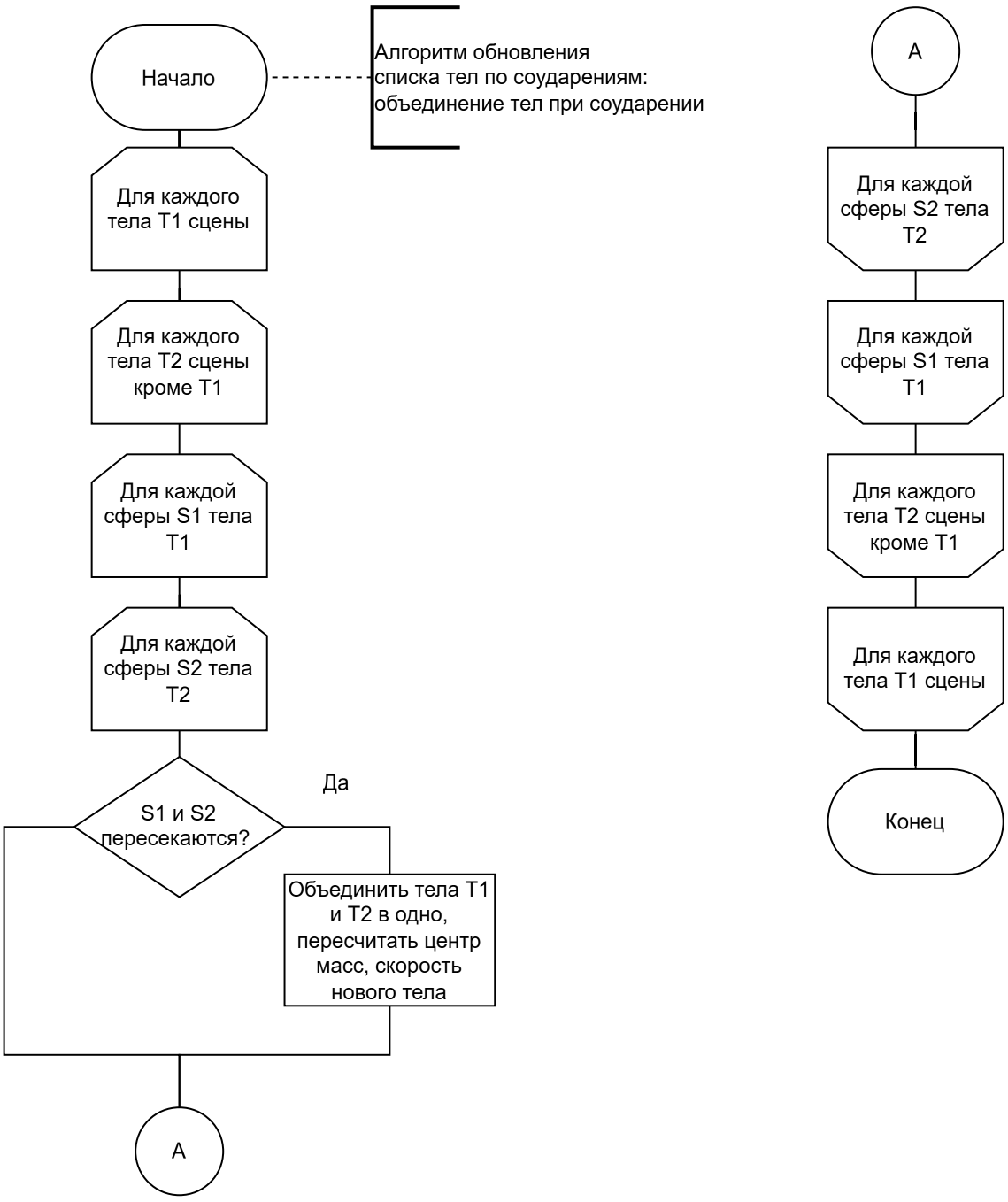


Рисунок 2.2 – Алгоритм расчета новых скоростей и позиций тел

## 2.3 Расчет параметров нового тела при столкновении

При абсолютно неупругом ударе два тела соединяются между собой, поэтому новое тело будет иметь массу равную сумме масс двух соударившихся тел. Так как соударение абсолютно неупругое, суммарный импульс тел сохраняется [3], поэтому для тел тел массами $m_1$ и $m_2$ со скоростями $\vec{V_1}$ и $\vec{V_2}$:

$$m_1 \cdot \vec{v_1} + m_2 \cdot \vec{v_2} = M \cdot \vec{V} \tag{2.5}$$

где M — суммарная масса нового тела, равная сумме масс $m_1$ и $m_2$, а $\vec{V}$ — вектор скорости нового тела.

Тогда, итоговая формула для скорости нового тела:

$$\vec{V} = \frac{m_1 \cdot \vec{v_1} + m_2 \cdot \vec{v_2}}{m_1 + m_2} \tag{2.6}$$

Центром масс (или центром инерции) системы материальных точек называется воображаемая точка $C$, положение которой характеризует распределение массы этой системы. Ее радиус-вектор равен [3]:

$$r_C = \frac{\sum_{i=1}^{n} m_i \cdot r_i}{\sum_{i=1}^{n} m_i} \tag{2.7}$$

Координатная форма:

$$P(x, y, z) = \begin{cases} x = \frac{\sum_{i=1}^{n} m_i \cdot x_i}{\sum_{i=1}^{n} m_i}, \\ y = \frac{\sum_{i=1}^{n} m_i \cdot y_i}{\sum_{i=1}^{n} m_i}, \\ z = \frac{\sum_{i=1}^{n} m_i \cdot z_i}{\sum_{i=1}^{n} m_i}, \end{cases} \tag{2.8}$$

## 2.4 Алгоритм Z-буфера

Так как на сцене присутствует источник света, для реалистичности полученной симуляции требуется отрисовывать тени, поэтому алгоритм Z–буфера был модифицирован для учета теней в сцене, состоящей только из сферических объектов и одного точечного источника света.

На рисунках 2.3–2.4 изображены схемы алгоритмов модифицированного Z–буфера и определения нахождения точки в тени.

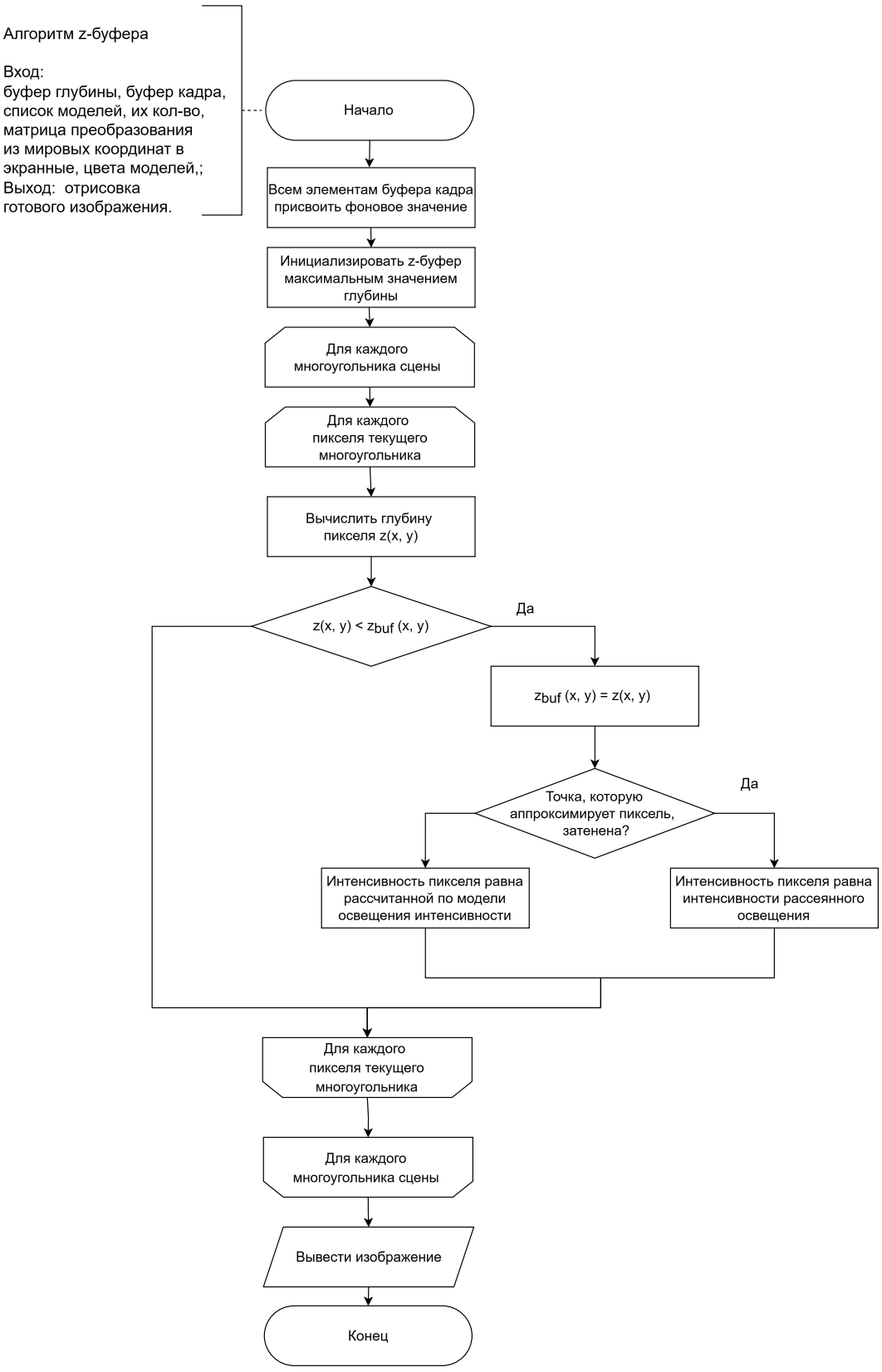


Рисунок 2.3 – Модифицированный алгоритм Z–буфера

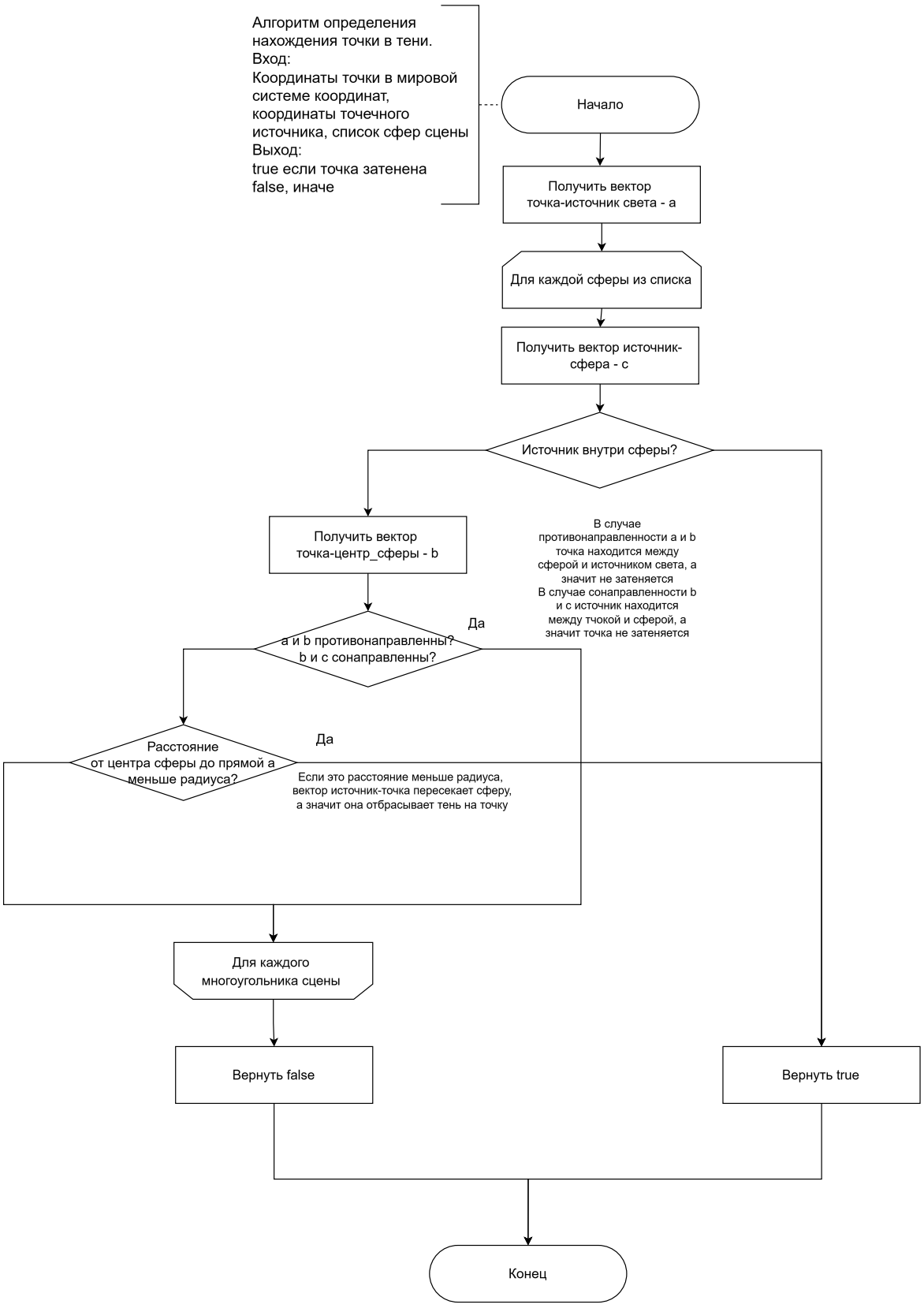


Рисунок 2.4 – Алгоритм определения затенения точки

## Вывод

В данном разделе были разработаны выбранные в аналитическом разделе алгоритмы.

# 3 Технологическая часть

## 3.1 Выбор языка программирования и среды разработки

Для написания программы был выбран язык C++ [5], это обусловлено тем, что этот язык является знакомым мне из курса ООП, а также данный язык является объектно–ориентированным, что даст возможность представлять объекты сцены в виде классов, что упростит разработку.

При написании программы использовалась среда разработки QT Creator, так как она бесплатна, позволяет писать приложения на языке C++ и отлаживать их, предоставляет возможности по разработке удобного интерфейса для программы в сжатые сроки и использует автоматическую систему сборки qmake [6].

## 3.2 Структура реализуемых классов

На рисунках 3.1 — 3.5 представлена структура классов программы

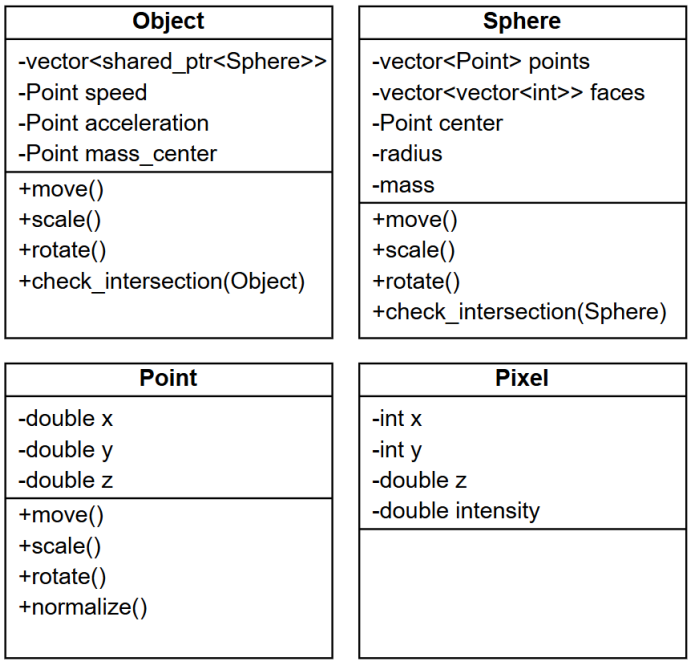


Рисунок 3.1 – Основные классы, реализующие объект

— Point — класс точки в пространстве, реализует хранение и геометрические преобразования;

— Pixel — класс пикселя;

— Sphere — класс сферы, хранит информацию о сфере в пространстве, а также ее полигональную модель в виде массива точек и массива граней;

— Object — класс состоящего из сфер объекта.

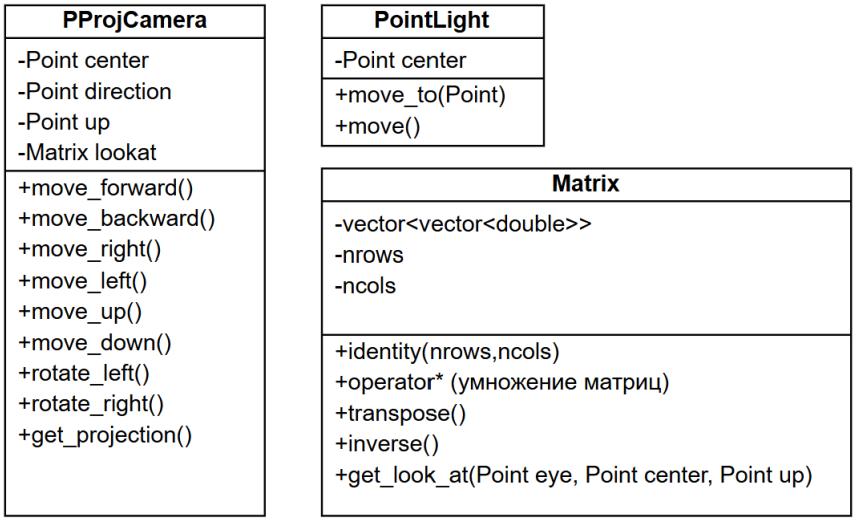


Рисунок 3.2 – Классы, реализующие камеру и свет

— PProjCamera — класс камеры перспективного проецирования, хранит информацию о положении камеры а также реализует логику преобразования мировых координат в экранные;

— PointLight — класс точечного источника света;

— Matrix — вспомогательный класс для камеры, реализует операции над матрицами и метод lookat, который по положению камеры возвращает матрицу преобразования мировых координат в систему координат камеры [4].

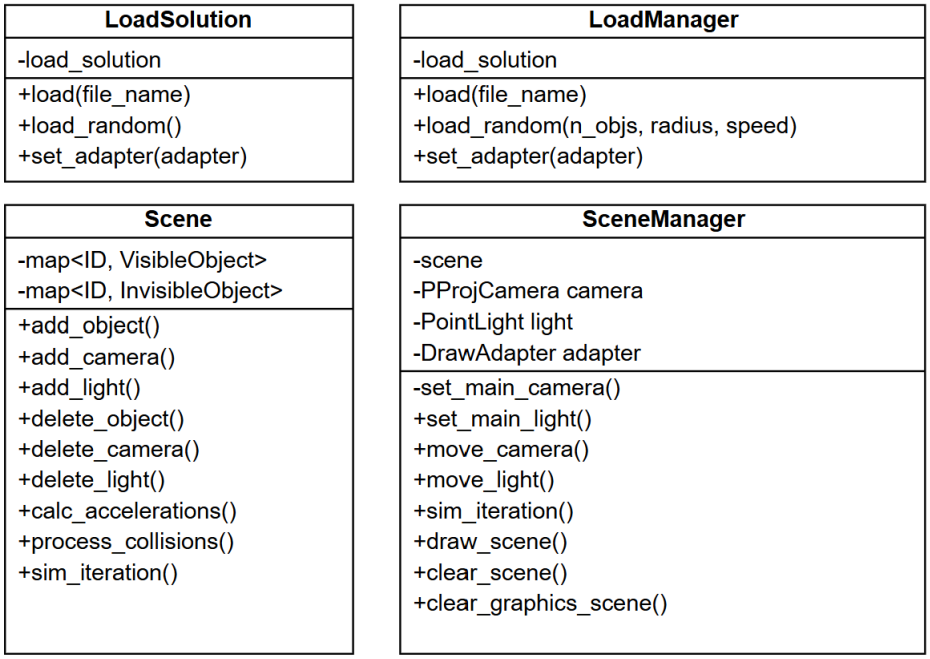


Рисунок 3.3 – Классы, реализующие сцену, и загрузку/генерацию сцены

— Scene — класс сцены, реализует всю физическую логику симуляции, хранит объекты, камеру и свет;

— SceneManager — класс посредник между сценой и пользователем;

— LoadSolution — класс, определяющий, как нужно загружать объекты сцены;

— LoadManager — менеджер загрузки/генерации сцены.

Текстовый файл загружаемой сцены содержит следующую информацию:

— первая строка —количество загружаемых сфер (натуральное число);

— последующие строки — описание сфер, по три строки на каждую;

— в первой строке три вещественных числа — координаты центра сферы по X, Y, Z;

— во второй строке содержатся еще три вещественных числа — координаты вектора начальной скорости, задаваемой сфере (относительно центра сферы);

— в третьей строке содержится вещественное число — радиус сферы.

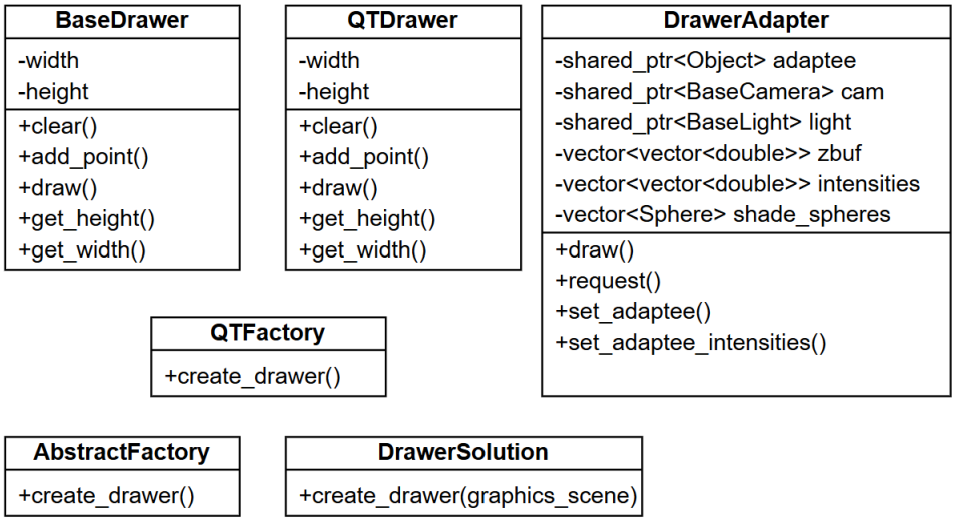


Рисунок 3.4 – Классы, реализующие логику рисования

— AbstractFactory, QTFactory — классы, создающие абстрактный отрисовщик и отрисовщик под библиотеку QT;

— BaseDrawer, QTDrawer — классы отрисовщиков, QTDrawer наследуется от BaseDrawer, как и любые другие конкретные отрисовщики, для быстрой смены графической среды (нет разницы на каком холсте рисовать, сцена от этого не зависит);

— DrawerSolution — класс, создающий отрисовщик в зависимости от поданной на вход графической сцены/холста;

— DrawerAdapter — класс, реализующий всю основную логику рисования сцены на любом холсте.

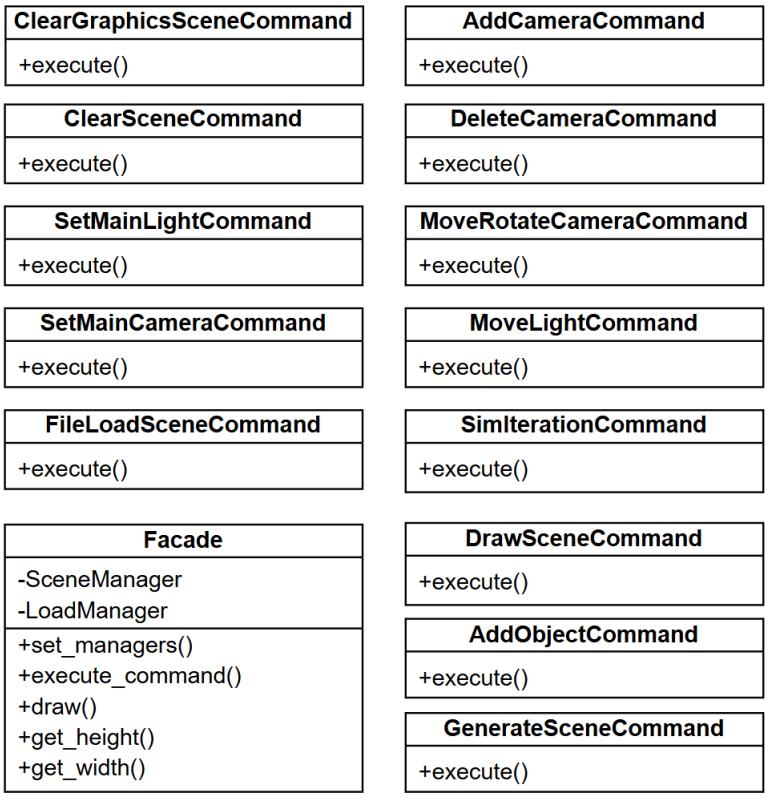


Рисунок 3.5 – Классы, реализующие связь между интерфейсом и менеджерами

— Facade — посредник между менеджерами и интерфейсом пользователя, выполняет любую из Command поданных на вход;

— Command — команды, выполняющие различные действия, переводят действие из интерфейса в команду для менеджеров.

## 3.3 Интерфейс программы

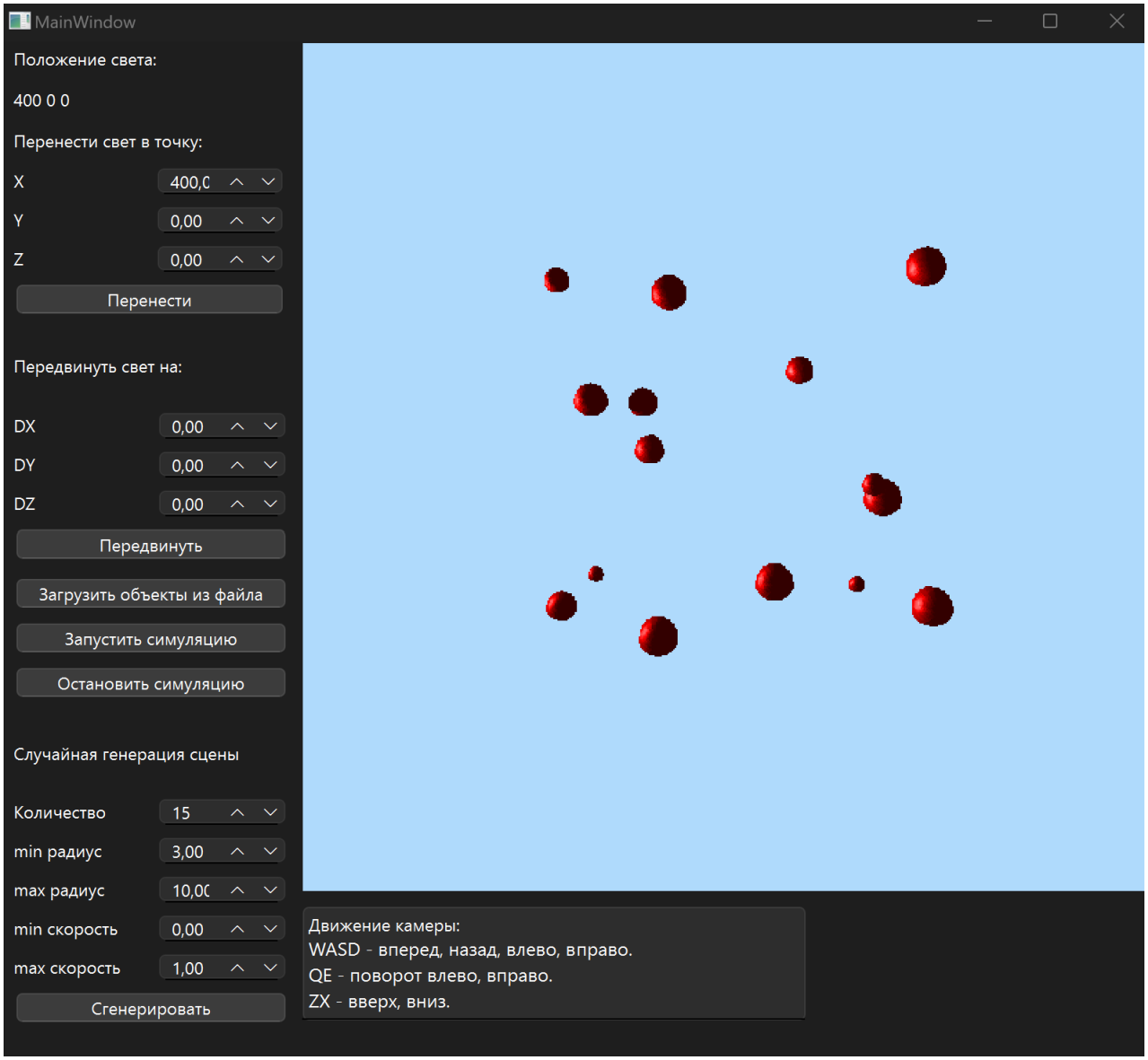


Рисунок 3.6 – Интерфейс программы

Интерфейс программы позволяет:

— загрузить сцену из правильно сформированного текстового файла;

— сгенеририровать случайную сцену по введенным: количеству объектов, минимальному и максимальному радиусу сферы, минимальному и максимальному модулю скорости (параметры сфер будут находится в этих заданных пределах);

— двигать и поворачивать камеру с помощью кнопок клавиатуры;

— двигать источник света в указанную точку или на указанный вектор.

# 4 Исследовательская часть

В данном разделе приведены технические характеристики устройства, на котором проводились замеры времени генерации кадра, а также результаты замеров

## 4.1 Характеристики устройства

Технические характеристики устройства, на котором проводилось исследование:

— Операционная система: Windows11 Домашняя, версия 23H2, сборка ОС 22631.4037;

— Оперативная память: 16 ГБ;

— Процессор: 13th Gen Intel(R) Core(TM) i5-13500H 2.60 ГГц.

При тестировании ноутбук был включен в сеть электропитания и нагружен только встроенными приложениями окружения, а также системой тестирования.

## 4.2 Результаты исследования

Для исследования зависимости времени генерации одного кадра от количества объектов на сцене были подготовлены несколько пробных сцен, включающих от 2 до 13 сфер одинакового радиуса.

Замеры были проведены для меридианных сфер построенных на: 10 параллелях и 10 меридианах, такая сфера содержит 180 полигонов; 20 параллелях и 20 меридианах, такая сфера содержит 760 полигонов.

Источник света помещался в середину сцены в точку с координатами (0, 0, 0) для идентичной освещенности всех сфер сцены, вокруг него равномерно вкруг выстраивались сферы. Каждой сфере задавалась начальная скорость, вектор которой был направлен от центра сцены к центру сферы.

На рисунке 4.1 изображена тестовая сфена из 10 сфер:

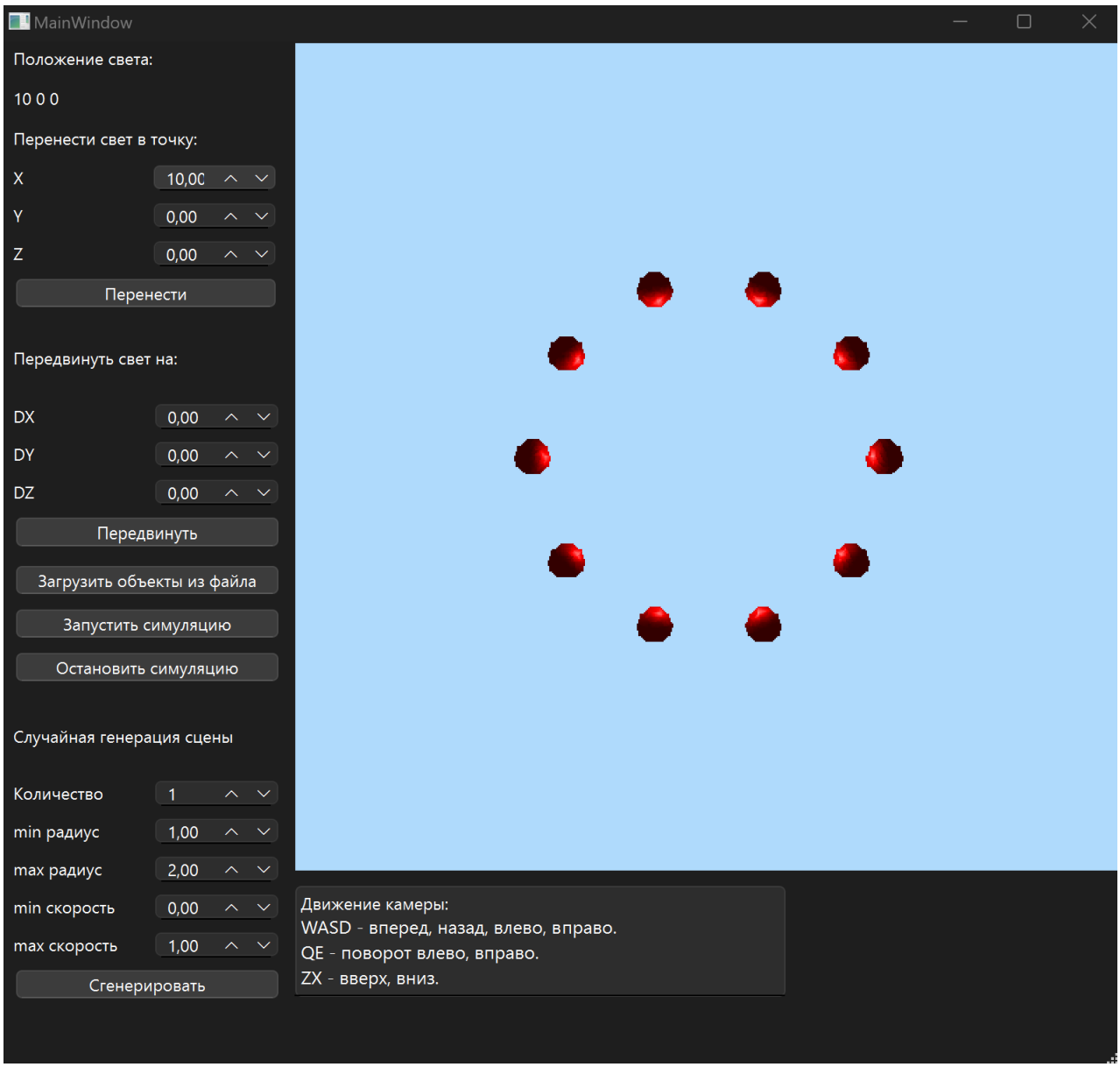


Рисунок 4.1 – Тестовая сцена

Результаты проведенного исследования представлены на графике 4.2. Аппроксимирующие полиномы:

— $0.00011 \cdot n^2 + 0.00028 \cdot n + 0.0175$, для сфер 20x20;

— $1^{-6} \cdot n^2 + 0.00068 \cdot n + 0.0143$, для сфер 10x10;

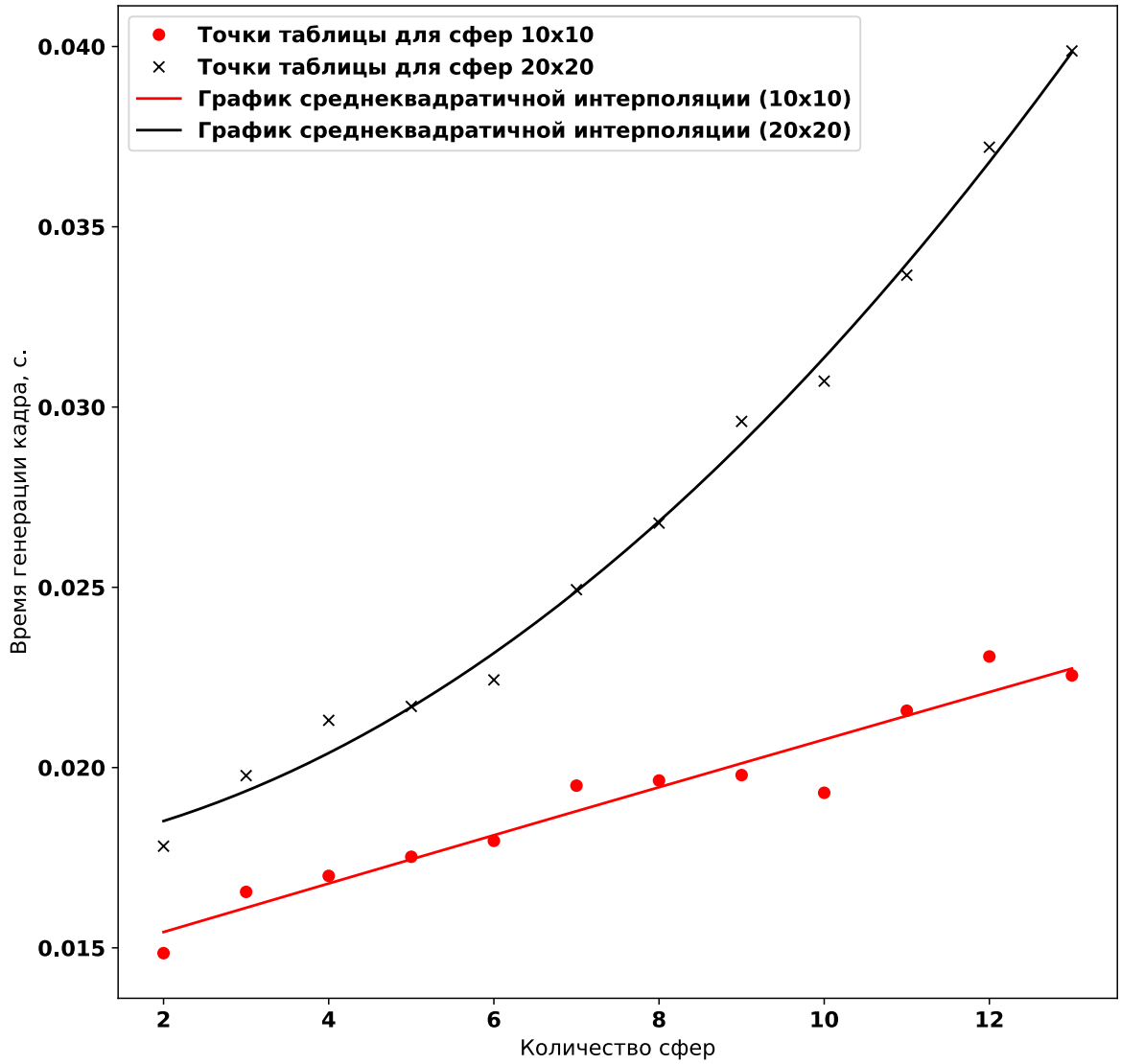


Рисунок 4.2 – Результаты измерений

Для оценки зависимости времени генерации кадра от количества сфер в сцене был использован алгоритм наилучшего квадратичного приближения при степени аппроксимирующего полинома, равной 2. Была выбрана именно такая степень, так как большинство функций, использованных при генерации кадра имеют квадратичную сложность, например алгоритм определения соударений, алгоритм определения сил взаимодействия и определения ускорений тел, алгоритм определения затенения точек.

Из проведенного исследования можно сделать вывод, что время генерации кадра квадратично зависит от количества объектов сцены, но при этом для менее детализированных (Low poly) сфер зависимость, на исследуемых количествах сфер, практически вырождается в линейную.

Результаты исследования совпали с ожидаемыми: время генерации кадра увеличивается с увеличением количества объектов сцены, но чем меньше полигонов в объектах, тем медленнее увеличивается это время.

## Вывод

В данном разделе было проведено исследование характеристик программы, описан интерфейс.

# ЗАКЛЮЧЕНИЕ

Цель данной курсовой работы была достигнута, была разработана программа, позволяющая моделировать движение группы сферических тел в пространстве под действием их собственного гравитационного поля. Программа предоставляет возможности по изменению положения камеры, источника света, генерации случайной сцены по заданным пользователем параметрам, а также позволяет загрузить сцену из правильно сформированного текстового файла.

Для достижения поставленной цели были решены следующие задачи:

— Проведен анализ существующих алгоритмов компьютерной графики;

— Выбраны подходящие для данной работы алгоритмы;

— Выбран язык программирования и среда разработки;

— Выбранные алгоритмы реализованы в виде одной программы с графическим интерфейсом;

— Проведено исследование быстродействия разработанного ПО в зависимости от количества объектов сцены.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1. Методы трехмерного моделирования [Электронный ресурс]. Режим доступа: https://studopedia.ru/19_307536_metodi-trehmernogo-modelirovaniya-karkasnoe-modelirovanie-poverhnostnoe-tverdotelnoe-modelirovanie-tipi-poverhnostey-chto-predstavlyayut-s-soboy-trehmernie-ob-ekti.html?ysclid=m1nj0jiox2234438145 (дата обращения: 29.09.2024).

2. Роджерс Д. Р60 Алгоритмические основы машинной графики: Пер. с англ. — М.: Мир, 1989. — 512 с, ил. ISBN 5-03-000476-9

3. Трофимова Т. И. Т761 Курс физики: учеб. пособие для вузов / Таисия Ивановна Трофимова. — 11-е изд., стер. — М.: Издательский центр «Академия», 2006. — 560 с. ISBN 5-7695-2629-7

4. Placing a Camera: the LookAt Function [Электронный ресурс]. Режим доступа: https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/lookat-function/framing-lookat-function.html (дата обращения: 20.10.2024).

5. Документация по языку C++ [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/ru-ru/cpp/cpp/?view=msvc-160 (дата обращения: 29.10.2024).

6. Manual qmake [Электронный ресурс]. Режим доступа: https://doc.qt.io/qt-6/qmake-manual.html (дата обращения: 30.10.2022).

# ПРИЛОЖЕНИЕ А

Презентация к курсовой работе содержит 13 слайдов.